Проектирование высоконагруженных систем на основе моделирования и теории массового обслуживания

Максим Юнусов





Предположим

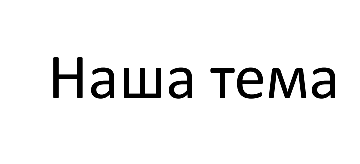
Вы проектируете некоторую высоконагруженную (высоконадежную и безопасную) систему.

Какие качества интересуют заказчика ?



Модные -ility

- Производительность
  - Время отклика
  - Пропускная способность



- Надежность
- Отказоустойчивость
- Безопасность
- …

- Скорость выхода на рынок (t2m)



Ограничения

- Люди
- Деньги
- Время

} Всего мало

- Непременно Agile (куда без него)



Решение в лоб («Русский водопад»)

- Анализ
  - Требования к производительности
  - Аппаратные и программные ограничения
  - Нагрузка
- Проектирование
  - Зачастую выбор референсной архитектуры
- Реализация
- Тестирование
  - Нагрузочное и т. п.
- Вывод в эксплуатацию
  - Мониторинг производительности



Решение в лоб («Русский водопад»)

- Анализ
  - Требования к производительности – на уровне интуиции
  - Аппаратные и программные ограничения
  - Нагрузка – либо сильно недооценена, либо с запасом
- Проектирование
  - Зачастую выбор референсной архитектуры – по предыдущему успешному проекту, либо «а давайте попробуем что-нибудь новенькое»
- Реализация – без оглядки на производительность либо оптимизируем все что видим
- Тестирование
  - Нагрузочное и т. п. – обычно слишком поздно (не готовы стенды, кейсы, интеграция)
- Вывод в эксплуатацию
  - Мониторинг производительности – и тут приходит осознание, что надо бы все переделать



Честный Agile (эволюционное проектирование)

- Анализ
- Проектирование
- Реализация
- Тестирование производительности (в рамках CI)
  - Нагрузочное и т. п.

Спринт
(1-2 недели)



«Честный» Agile (эволюционное проектирование)

- Анализ
- Проектирование
- Реализация
- Выравниваемся с другими командами
- Ждем разворачивания стендов
- Тестирование производительности (в рамках CI)
  - Нагрузочное и т. п.

Релиз (1-2 месяца)



Любимый подход менеджера ("fix-it-later")

В основной части цикла разработки игнорируйте вопросы эффективности

Займитесь производительностью, когда ваша программа заработает корректно и проект будет отражать ваше полное понимание того, как должен быть структурирован код программы

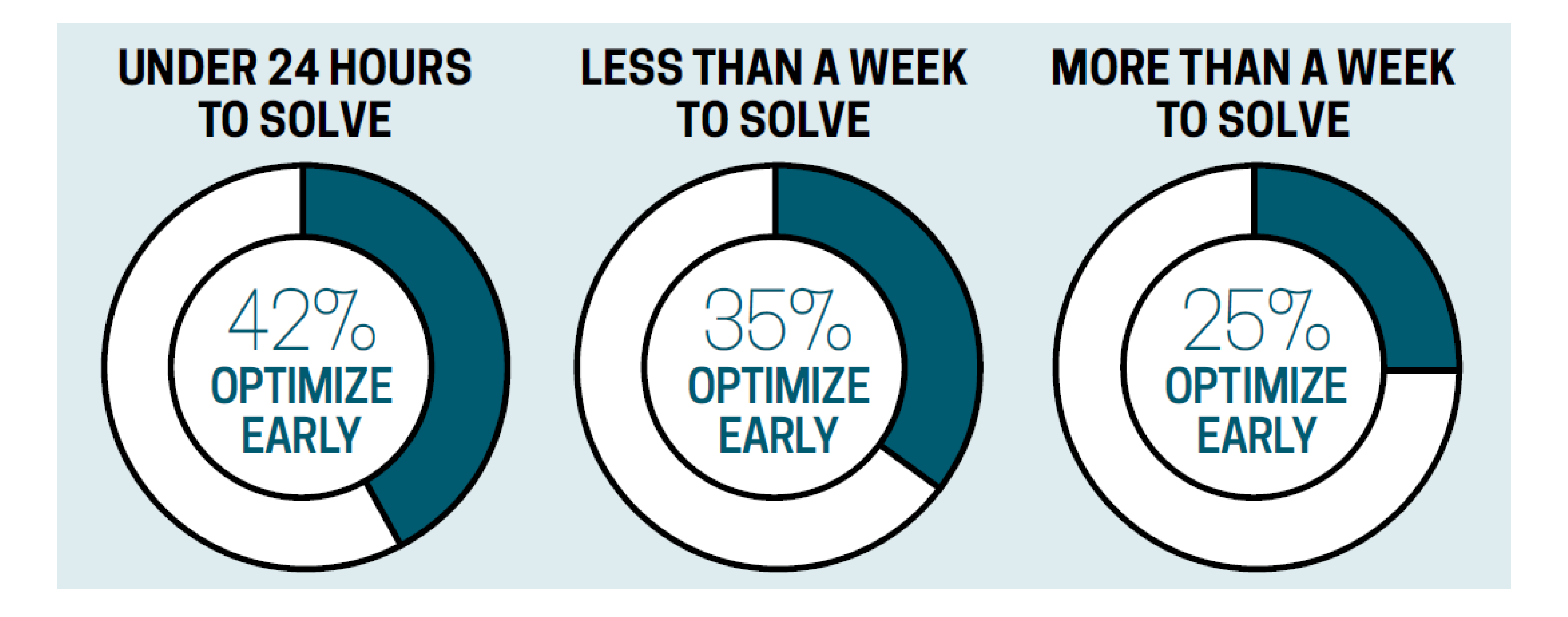




Оуэр (Auer) и Бек (Beck) , 1996



Результат (по 400+ IT-компаниям)



dzone's 2015 guide to performance and monitoring

Решение

Проактивный подход

- Устраняем проблему **до** ее возникновения



Полная цитата

В основной части цикла разработки игнорируйте вопросы эффективности

Займитесь производительностью, когда ваша программа заработает корректно и проект будет отражать ваше полное понимание того, как должен быть структурирован код программы

Вносите в проект лишь такие изменения, которые могут указать дорогу к возможностям его улучшения





Кен Оуэр (Ken Auer) и Кент Бек (Kent Beck), 1996



И еще одно мнение

Чтобы говорить о производительности, **я предпочитаю увидеть работающую систему**, измерить ее характеристики и, учитывая полученные результаты, применить строго определенные процедуры оптимизации



Martin Fowler



И еще одно мнение

Чтобы говорить о производительности, **я предпочитаю увидеть работающую систему**, измерить ее характеристики и, учитывая полученные результаты, применить строго определенные процедуры оптимизации

- Однако, некоторые архитектурные решения определяют параметры производительности таким образом, что устранение возможных проблем средствами оптимизации затрудняется
- ***Именно поэтому к принятию таких решений всегда стоит подходить с большой ответственностью***



Martin Fowler



Как?

- У мэтров нет ответа на вопрос «какие именно решения нужно принимать предварительно»
- Как не скатиться к преждевременной оптимизации, которая, как известно, «корень всех зол в программировании»

Donald Knuth

- Попробуем позаимствовать инструменты программной инженерии
  - **Математическое моделирование**



Мнение

- **Построение моделей производительности — дорогостоящий процесс ввиду их сложности**



Проверено на практике

- Очень простые модели могут предоставить информацию, необходимую для идентификации проблем производительности и оценки альтернатив, позволяющих их решить
- Зачастую построение модели занимает **несколько минут**



Демонстрация

- Возьмём для примера очень распространённую задачу – построение конвейера



Формирование печатных представлений

сообщения
1000 mps
(в перспективе 1000000 mps)

Прием
и регистрация

Канонизация

Обогащение

Сортировка
и группировка

Генерация
печатного
представления

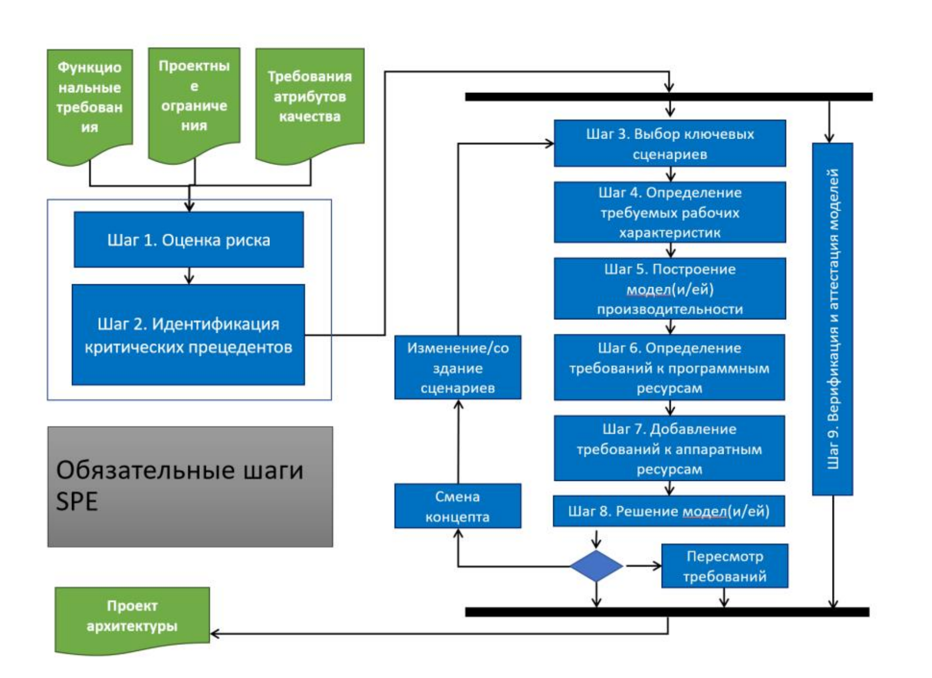
EXCALIDRAW

Требование заказчика:

Сообщение должно быть обработано за 1 сек



Пробуем пойти проторенным путем (методология SPE)



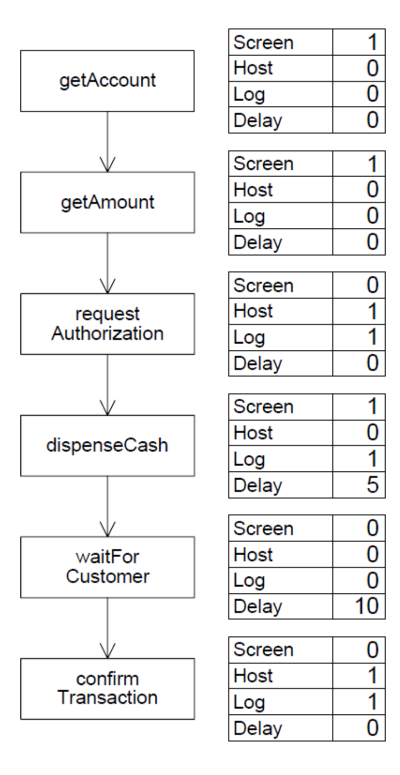
Software **P**erformance **E**ngineering

- Контроль производительности на всех этапах разработки
- Возможность определения узких мест системы **до** построения системы
- Возможность оценить альтернативные варианты решения без необходимости реализовать каждое



Шаги SPE (упрощенно)

- Выявляем проблемный с т. з. требований и нагрузки сценарий
- Модель работы программы
  - Декомпозируем сценарий, разбивая его на этапы
  - Оцениваем время выполнения каждого этапа
    - Оптимистично
    - Пессимистично
  - Если на этом этапе модель показывает неудовлетворительный результат, меняем архитектуру
- Модель работы системы
  - Добавляем нагрузку и рассчитываем основные характеристики модели (время отклика, утилизация и т. п.)
  - Находим бутылочное горлышко
  - Устраняем проблему, применяя различные паттерны (или тактики)

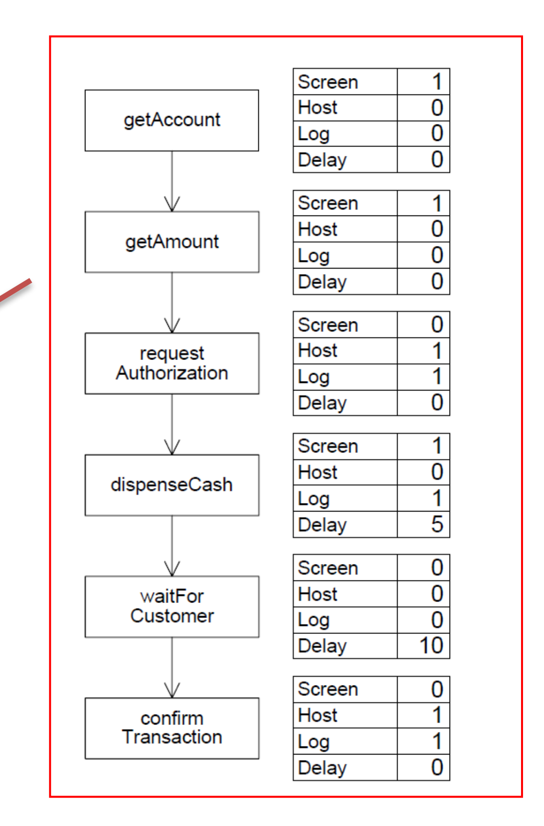




Шаги SPE (упрощенно)

- Выявляем проблемный с т. з. требований и нагрузки сценарий
- Модель работы программы
  - Декомпозируем сценарий, разбивая его на этапы
  - Оцениваем время выполнения каждого этапа
    - Оптимистично
    - Пессимистично
  - Если на этом этапе модель показывает неудовлетворительный результат, меняем архитектуру
- Модель работы системы
  - Добавляем нагрузку и рассчитываем основные характеристики модели (время отклика, утилизация и т. п.)
  - Находим бутылочное горлышко
  - Устраняем проблему, применяя различные паттерны (или тактики)

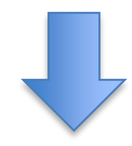
КАК ?





Отход от плана

- Оцениваем время выполнения каждого этапа



- Бюджетируем выполнение каждого этапа
  – Бюджет для своих компонентов
  – Предположения для внешних компонентов и устройств



Что дальше?

- Передаем бюджеты разработчикам как требования по производительности
  - Отслеживать выполнение этих требований можем на уровне интеграционных тестов раз в 15 минут)
  - Используем микробенчмарки
- Предположения включаются в контракты с внешними командами и в ТЗ



Бюджеты и предположения

сообщения
1000 mps
(в перспективе 1000000 mps)

10 ms

Прием и регистрация

100 ms

Канонизация

2 ms на передачу по сети

Обогащение

100 ms

50 ms

сортировка и группировка

200 ms

500 ms
3 ТБ

Генерация печатного представления

EXCALIDRAW
Sketch handdrawn like diagrams

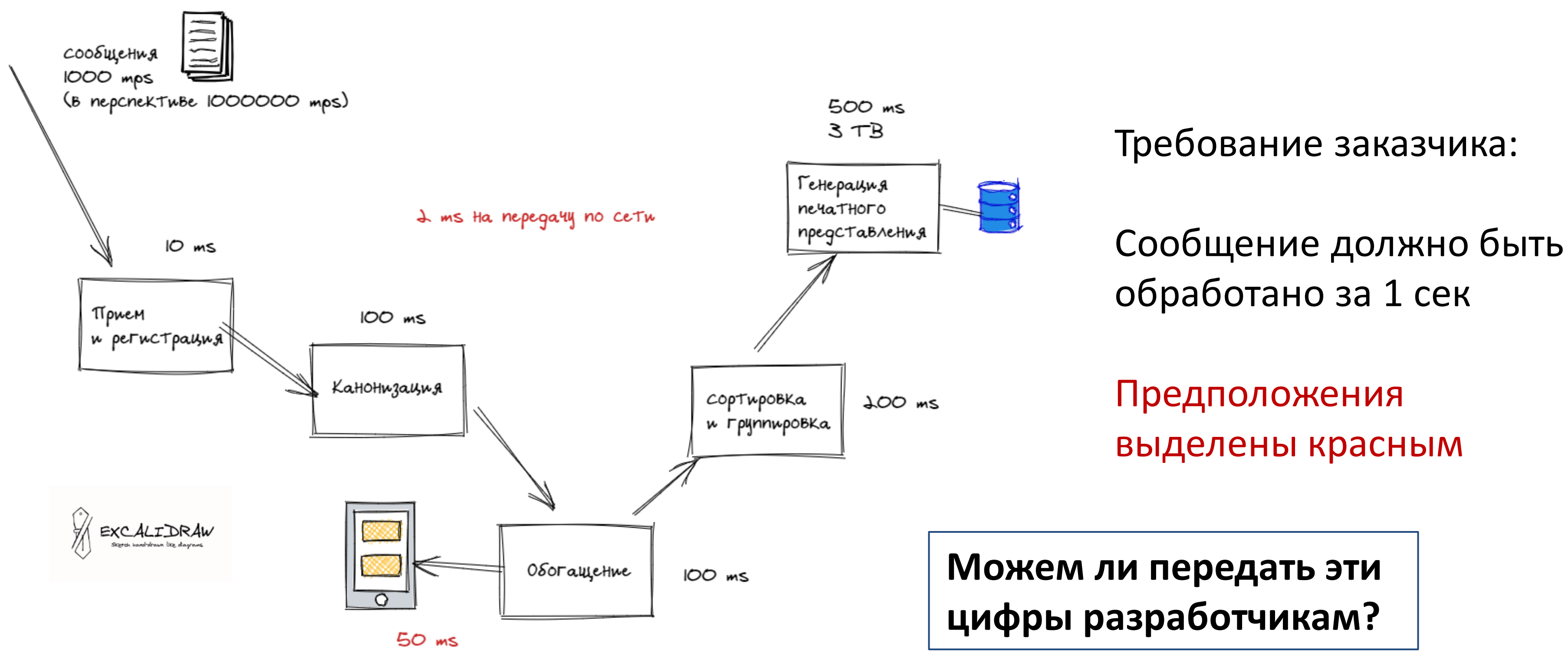
Требование заказчика:

Сообщение должно быть обработано за 1 сек

Предположения выделены красным



Бюджеты и предположения



Требование заказчика:

Сообщение должно быть обработано за 1 сек

Предположения выделены красным

Можем ли передать эти цифры разработчикам?

Учет нагрузки

- Предыдущая модель бюджетирует общее время пребывания на узле
- Это время не может быть использовано в контрактах, так как большую часть времени сообщения будут терять не на обработке, а **простаивая в очередях**
  - **Длина очереди зависит от нагрузки**
  - Требуется учет нагрузки



Учет простоя в очередях

Средняя длина очереди
(Queue length)

$$N = \frac{U}{1 - U}$$

Среднее время ожидания
(Wait time)

$$WT = \frac{U}{1 - U} \cdot S$$

Среднее время пребывания на узле
(Residence time)

$$RT = WT + S$$

$$RT = \frac{S}{1 - U}$$

$$RT = \frac{S}{1 - S \cdot X}$$

Где,
U – утилизация ресурса
X – нагрузка (операции в секунду)
S – время обслуживания на узле (тот самый бюджет)

Закон утилизации

$$U = S \cdot X$$



Обратная формула

- Время обработки через время пребывания

$$RT = \frac{S}{1 - S \cdot X} \rightarrow S = \frac{RT}{1 + RT \cdot X}$$

- Например, для компонента «генерация печатного представления» время пребывания RT = 500 ms (бюджет)
  – Время обслуживания S ≈ **0.998 µs**, при X = 1.000 mps
  – Время обслуживания S ≈ **0.999 ns**, при X = 1.000.000 mps



Очевидное решение

- Масштабируем
  - Например, выделив под задачу 1000 ядер (1000 pod сервиса генерации печатного представления), снижаем нагрузку до 1 mps на ядро
  - Время обслуживания на ядре S ≈ **333 ms**



Очевидное решение

- Масштабируем
  - Например, выделив под задачу 1000 ядер (1000 pod сервиса генерации печатного представления), снижаем нагрузку до 1 mps на ядро
  - Время обслуживания на ядре S ≈ **333 ms**

Можем ли передать эту цифру разработчикам ?



Проблема усреднения

На самом деле ситуация еще хуже

- Приведенные выше формулы работают со **средними** величинами
- Заказчику же хочется, чтобы **большинство** сообщений (или все) обрабатывались вовремя
- Упс! Мы не учли это при фиксации требований
  - **Переписываем требования**:
    99.5 % сообщений должно быть обработано за **1 сек**.

Нужен перерасчет бюджета



Связь среднего и перцентили

Многие системы могут быть аппроксимированы формулой

$$R_x = R \ln\left(\frac{1}{1-x}\right)$$

То есть,

$$R_{50} \approx 0.7R \quad (\text{медиана})$$
$$R_{95} \approx 3R$$
$$R_{99} \approx 4{,}6R$$
$$R_{995} \approx 5{,}3R$$
$$R_{999} \approx 6.9R$$
$$R_{99999} \approx 11.5R$$



Учет распределения

- $RT_{995} = 500\ ms \approx 5{,}3R \rightarrow R \approx \mathbf{90\ ms}$
  - Время обслуживания S ≈ **82 ms**, при X = 1.000 mps, «размазанных» по 1.000 ядер

82 ms вместо ранее рассчитанных 300 ms



Промежуточный итог

Что мы сделали:

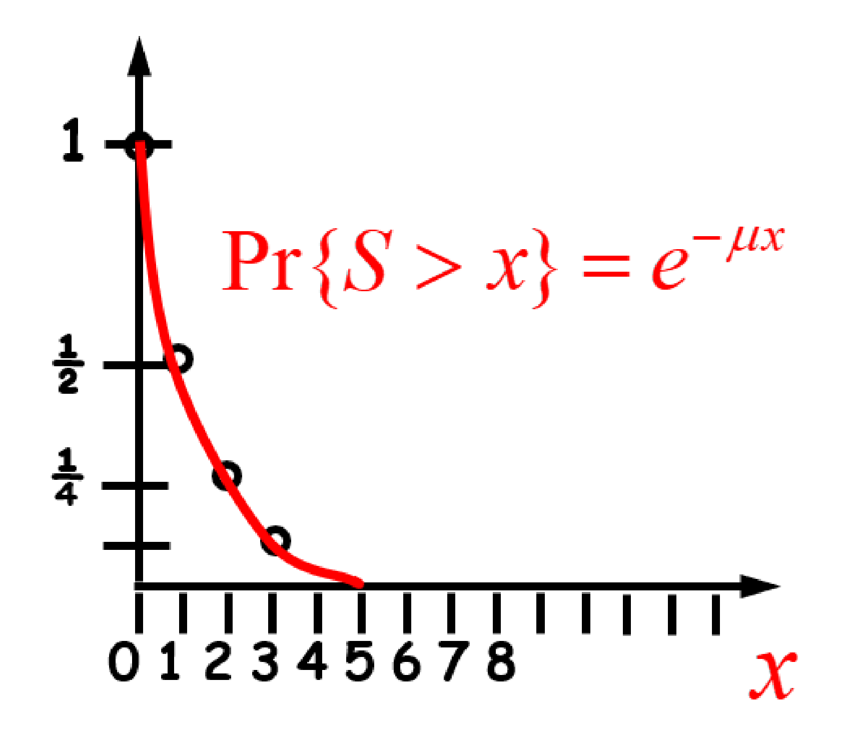
- Мы уточнили у заказчика ожидаемое время выполнения одного сообщения **с учетом перцентилей**
- Распределили это время по компонентам (как время пребывания, то есть обслуживания плюс простой в очередях)
- Используя полученные знания, вывели из времени пребывания время обслуживания ($S = 82$ ms)

Можем ли передать эту цифру разработчикам ?



Нюансы

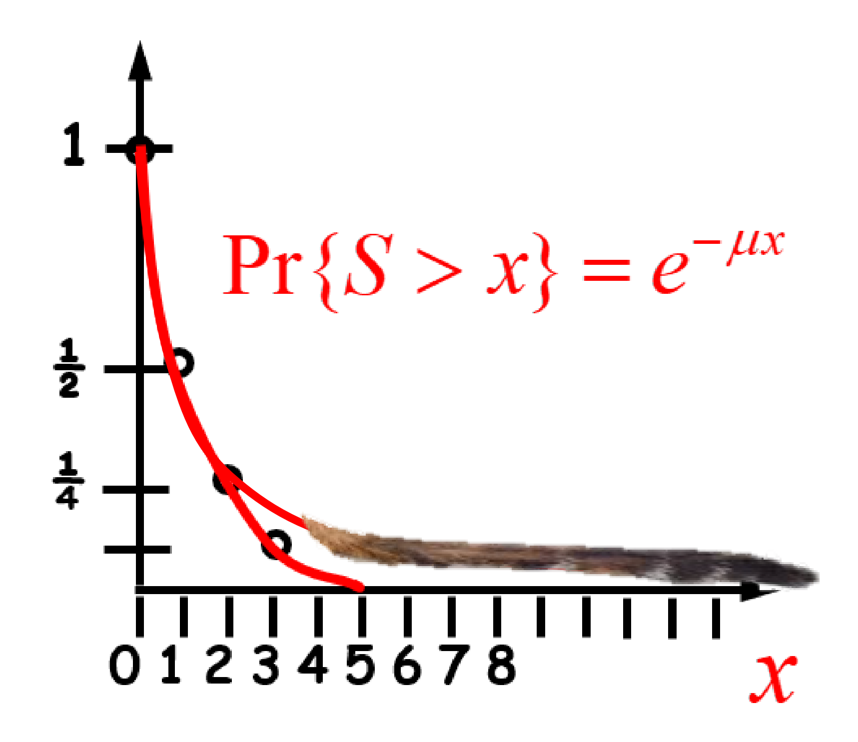
- Приведенные выше формулы работают для очередей вида M/M/1
(где M – Markovian или memoryless)
  - Время прибытия и время обработки варьируется экспоненциально
- Это достаточно распространённый, **но не общий** сценарий





Нюансы

- Приведенные выше формулы работают для очередей вида M/M/1
  (где M – Markovian или memoryless)
  - Время прибытия и время обработки варьируется экспоненциально
- Это достаточно распространённый, **но не общий** сценарий
  - Например, может быть очень солидный «хвостик»





Более общий случай (M/G/1)

Формула Поллачека-Хинчина
(Pollaczek-Khintchine formula)

$$RT = \frac{1}{1-U} \cdot \frac{E[S^2]}{2E[S]}$$

Время отклика существенно зависит как от *утилизации,* так и от *вариативности*

Увеличение утилизации в два раза может привести к увеличению времени ожидания от **4 до ∞ раз**

Низкая утилизация не всегда приводит к малому времени ожидания в очередях!

Наш случай

- Часть сообщений (95%) превращаются в печатную форму за 50 ms
- Часть сообщений (5 %) обрабатываются за 2 sec
- Среднее время пребывания $RT = \frac{1}{1-U} \cdot \frac{E[S^2]}{2E[S]}$

X = 1 mps	
95%	S = 0.05 sec
5%	S = 2 sec

$$\mathrm{E}[S] = 0.95 \times 0.05 + 0.05 \times 2 \approx 0.15\ sec$$
$$\mathrm{E}[S^2] = 0.95 \times 0.05^2 + 0.05 \times 2^2 \approx 0.2\ sec^2$$
$$U = 1 \times 0.15 \approx 0.15\ (15\%)$$
$$\boldsymbol{RT \approx 0.6\ sec}$$



Пятиминутная автоматизация

G2 | = (1 / 1 - A2 * D11) * (E11/(2 *D11))

	A	B	C	D	E	F	G
1	X	P	S	E[S]	E[S^2]		RT
2	1	0,95	0,05	0,0475	0,002375		0,58482945
3		0,05	2	0,1	0,2		
4				0	0		
5				0	0		
6				0	0		
7				0	0		
8				0	0		
9				0	0		
10				0	0		
11		1		0,1475	0,202375		
12							



Работа с вариативностью

- Вариативность времени обработки сказывается на среднем времени ожидания очень неприятным образом
- Можно учитывать длинные хвосты в своих моделях, но я зачастую иду другим путем
- Разделяем сервисы, обрабатывающие долгие задачи и сервисы, работающие над короткими задачами,
  - существенно улучшаем производительность
  - упрощаем расчеты



В итоге

- Промоделировали проектируемую систему (в Microsoft Excel)
- Определили вопросы для уточнения требований (перцентили)
- Использовали пару архитектурных тактик
- Получили на руки цифры для обсуждения
  - бюджет для разработчиков (сдюжат или нет)
  - предположения и контракты для внешних команд
  - сайзинг для бизнеса

- **Уложились в 40 минут**



В дальнейшем

- Получая первые результаты реализации, модифицируем нашу модель (т. н. «Стратегия адаптируемой точности»)
- Выявляем узкие места в системе (где самая длинная очередь)
  - Применяем тактики, позволяющие передвинуть бутылочное горлышко на другой участок



Список применяемых тактик

- Собран из разных источников
- Упорядочен и приоритизирован
  – от самых простых и дешевых до сложных и дорогих, приводящих к деградации других качеств системы

- Тема для отдельного разговора



Честный Agile (эволюционное проектирование)

- Анализ
- **Построение модели производительности**
- Проектирование
- Реализация
- Тестирование производительности (в рамках CI)
  - Нагрузочное и т. п.

Спринт
(1-2 недели)

Стабильно!



Вопросы?

- Доступен и после конференции по ссылке



https://t.me/MaximYunusov



